

Starting R: Basics of Custom Graphics

Danny Kaplan

March 12, 2009

Customizing Graphics I

- ▶ We have been using the **lattice** graphics package. This must be initialized
 - > `library(lattice)`
- ▶ It's very powerful and it's not hard to do some things simply. But it becomes more involved when you want to do something custom.
- ▶ Here are some examples, using the “wages.sav” data set. The point is not to show you about wage structures, but how to produce graphics.

The Wage Data

Data on 93 skilled, entry-level clerical workers hired by the Harris Trust and Savings Bank from 1969 to 1977. The data were made public and are described further in, Roberts, H. V., (1979). *Harris Trust and Savings Bank: An analysis of employee compensation*. Report 7946, Center for Mathematical Studies in Business and Economics, University of Chicago Graduate School of Business.

The data set includes the following variables,

- ▶ **salary**: Starting salary for the employee, in dollars
- ▶ **sex**: Sex of the employee (MALE or FEMALE)
- ▶ **senior**: Seniority of the employee, months since employee was hired
- ▶ **age**: Age of the employee at the time of hire, in months
- ▶ **educ**: Amount of education of the employee at the time of hire, in months
- ▶ **exper**: Amount of prior experience, in months

Reading in the Data and Fixing It I

```
> wages = read.spss("/Users/kaplan/kaplanfiles/Projects/R-U  
to.data.frame = TRUE)
```

```
> names(wages)
```

```
[1] "salary" "sex"      "senior" "age"      "educ"  
[6] "exper"
```

```
> summary(wages)
```

salary		sex			
Min.	:3900	FEMALE	:61		
1st Qu.	:4980	MALE	:32		
Median	:5400				
Mean	:5420				
3rd Qu.	:6000				
Max.	:8100				
senior		age		educ	
Min.	:65.00	Min.	:280.0	Min.	: 8.00

Reading in the Data and Fixing It II

```
1st Qu.:74.00    1st Qu.:349.0    1st Qu.:12.00
Median  :84.00    Median  :468.0    Median  :12.00
Mean    :82.28    Mean    :474.4    Mean    :12.51
3rd Qu.:90.00    3rd Qu.:590.0    3rd Qu.:15.00
Max.    :98.00    Max.    :774.0    Max.    :16.00
```

exper

```
Min.    : 0.0
1st Qu.: 35.5
Median  : 70.0
Mean    :100.9
3rd Qu.:144.0
Max.    :381.0
```

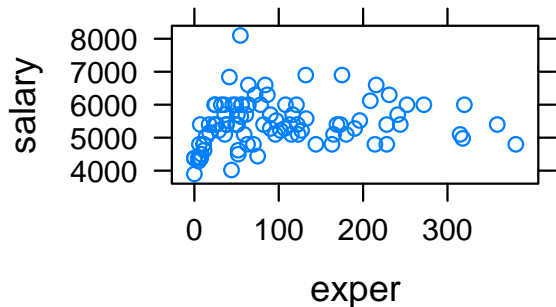
Unfortunately, the coding for sex includes blank characters. Let's fix this:

```
> wages$sex = factor(wages$sex, labels = c("F",
      "M"))
```

A Basic Scatter Plot I

I'll call the plots with names like `plot1`, `plot2`, and so on

```
> plot1 = xyplot(salary ~ exper, data = wages)
```



Changing Plots in the Lattice System

You will often want to change labels, colors, symbols, etc. from their defaults.

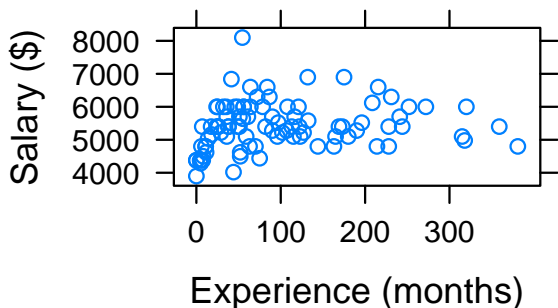
Two basic ways to do this:

- ▶ Pass the changes as arguments in the original statement making the graphic.
- ▶ Use the update function.

Change the Labels I

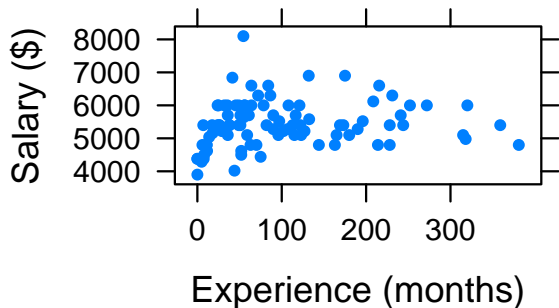
```
> plot1 = update(plot1, xlab = "Experience (months)",  
  ylab = "Salary ($)")
```

Note the re-assignment to plot1.



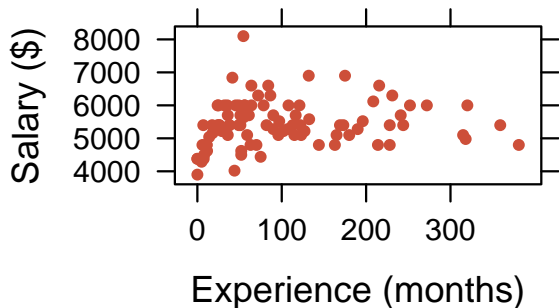
Change the Symbol I

```
> plot1 = update(plot1, pch = 20)
```



Change the Color I

```
> plot1 = update(plot1, col = "tomato3")
```



Colors I

- ▶ There are many named colors. The operator `colors` will list the available ones:

```
> head(colors())
```

```
[1] "white"           "aliceblue"  
[3] "antiquewhite"   "antiquewhite1"  
[5] "antiquewhite2"  "antiquewhite3"
```

- ▶ You can use `rgb` to create any color from an Red-Green-Blue specification.

```
> dannyColor = rgb(0, 0.59, 0.86, 0.3)
```

The last argument is optional: the transparency (“alpha”). 1 means opaque, 0 means completely transparent.

- ▶ You can see the RGB breakdown of a named color using `col2rgb`:

```
> col2rgb("tomato3")/255
```

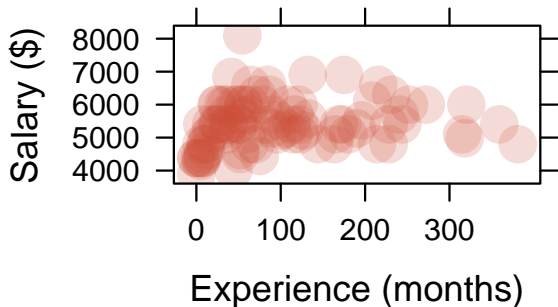
Colors II

```
          [,1]  
red      0.8039216  
green    0.3098039  
blue     0.2235294
```

Example: Using Transparency to Indicate Density I

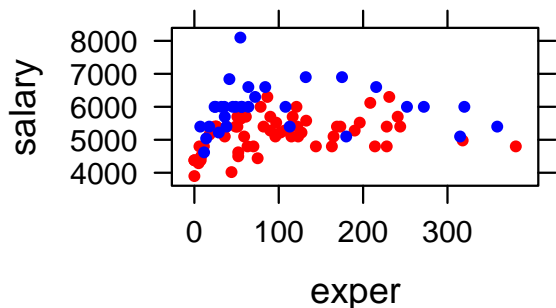
I'll make the scatter plot dots very big, and somewhat transparent:

```
> plot2 = update(plot1, cex = 3, col = rgb(0.804,  
      0.31, 0.223, 0.2))
```



Different Colors for the Different Sexes I

```
> plot3 = xyplot(salary ~ exper, groups = sex,  
  data = wages, col = c("red", "blue"),  
  pch = 20)
```

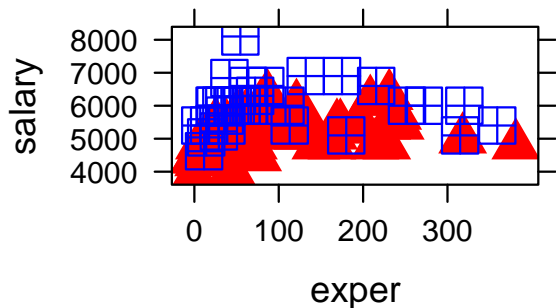


Different Symbols for Different Groups I

If the groups are broken out by a variable, use the groups facility in lattice:

```
> plot4 = xyplot(salary ~ exper, groups = sex,  
  data = wages, col = c("red", "blue"),  
  pch = c(17, 12), cex = 2)
```

Different Symbols for Different Groups II

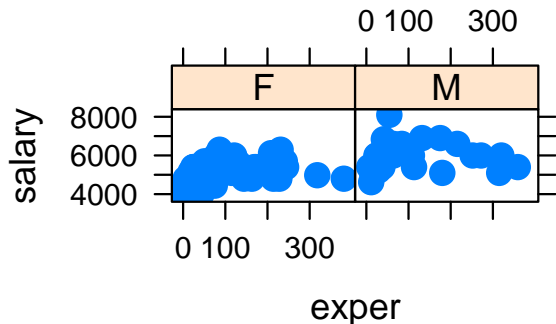


You can change the labels, colors, plotting symbols, etc. with `update` if you want.

Other Lattice Facilities I

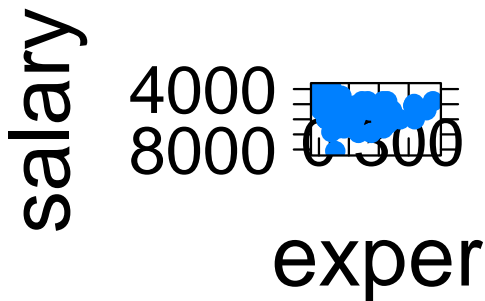
Lattice lets you draw different panels for different groups, etc.

```
> plot5 = xyplot(salary ~ exper | sex, data = wages,  
  pch = c(20), cex = 2)
```



Changing Font Sizes I

```
> plot5 = xyplot(salary ~ exper, data = wages,  
  pch = 20, par.settings = list(fontsize = list(text = 20,  
    points = 15)))
```



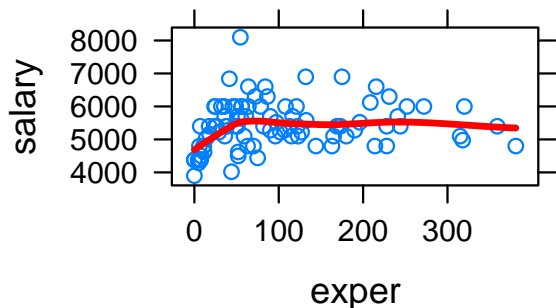
Adding Annotations to Plots I

You write a function that constructs the plot you want out of the basic **panel** functions and other basic graphics functions such as **llines**, **lpoints**, **ltext**. Then you pass your panel function to the graphics routines.

Example: Add a loess regression line to the scatter plot. I will define `myPanelFun` to use the built-in `panel.xyplot` and `panel.loess`.

```
> myPanelFun = function(x, y, ...) {  
  panel.xyplot(x, y, ...)  
  panel.loess(x, y, ..., col = "red",  
             lwd = 3)  
}  
  
> plot6 = xyplot(salary ~ exper, data = wages,  
                panel = myPanelFun)
```

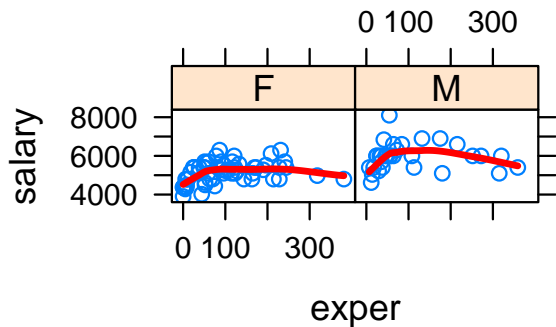
Adding Annotations to Plots II



This may seem like a lot of work just to add a loess line. But the function you create can be re-used. For instance:

```
> plot6 = xyplot(salary ~ exper | sex, data = wages,  
  panel = myPanelFun)
```

Adding Annotations to Plots III



Conditioning on Quantitative Variables I

Create a “shingle,” which splits the cases up according to their values of the variable:

```
> age.group = equal.count(wages$age, number = 4,
  overlap = 0)

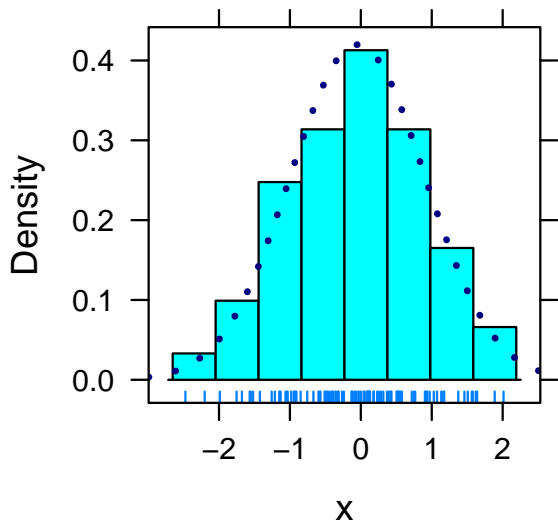
> linePanel = function(x, y, coefs, ...) {
  panel.xyplot(x, y, ...)
  panel.lmline(x, y, ..., col = "red",
    lwd = 3)
  panel.abline(a = coefs[1], b = coefs[2],
    lwd = 2, col = "blue")
}

> plot7 = xyplot(salary ~ exper | age.group +
  sex, data = wages, panel = linePanel,
  coefs = lm(salary ~ exper, data = wages)$coef)
```


Examples from Randy Pruim I

```
> x = rnorm(100)
> plot8 = histogram(~x, type = "density",
  panel = function(x, y, ...) {
    panel.rug(x, ...)
    panel.histogram(x, ...)
    panel.mathdensity(dmath = dnorm,
      args = list(mean = mean(x),
        sd = sd(x)), lwd = 3,
      col = "navy", lty = 3, ...)
  })
```

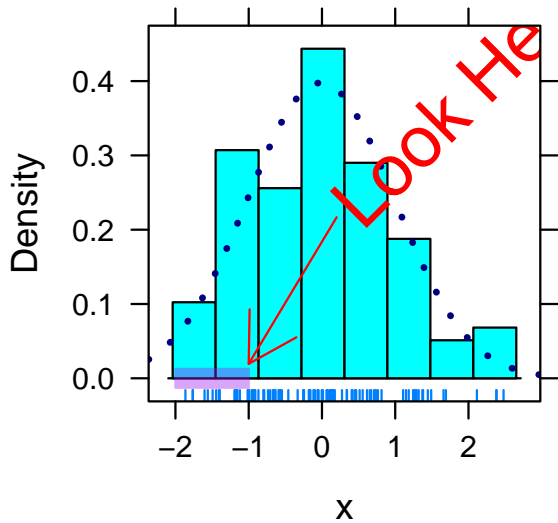
Examples from Randy Pruim II



The next one uses “grid” graphics: primitives for graphics that are in the grid library:

```
> library(grid)
> x = rnorm(100)
> demoPanel = function(x, y, ...) {
  panel.rug(x, ...)
  panel.histogram(x, ...)
  panel.mathdensity(dmath = dnorm, args = list(mean = me
    sd = sd(x)), lwd = 3, col = "navy",
    lty = 3, ...)
  grid.text("Look Here", x = 0.5, y = 0.5,
    just = "left", default.units = "npc",
    rot = 45, gp = gpar(col = "red",
      cex = 2))
  grid.segments(x0 = 0.48, x1 = unit(-1,
    "native"), y0 = 0.49, y1 = 0.1,
    arrow = arrow(), default.units = "npc",
    gp = gpar(col = "red"))
```

```
grid.rect(x = -2, y = 0, width = 1,  
          height = unit(0.05, "npc"), default.units = "native",  
          just = "left", gp = gpar(col = "purple",  
          fill = "purple", alpha = 0.4))  
}  
  
> plot9 = histogram(~x, type = "density",  
                    panel = demoPanel)
```



Work Flow

I build graphics incrementally and iteratively.

- ▶ Open an R source file to hold the graphics statements you will be writing.
- ▶ Try out a simple graphics command in a command window. When it is working, copy it over to the source file.
- ▶ Try out the incremental improvements in the graph in the command window. As you get these working, copy them over to the source file.
- ▶ The graphic being displayed will “evolve” as you add in more commands. You will occasionally want to start over. Do this by “sourcing” the R source file to regenerate the graphic to the point that you are satisfied.